Experiences with the AT&T Pixel Machine at NAS

G. David Kerlick[1]

Report RNR-89-001, March 1989

Computer Sciences Corporation
NASA Ames Research Center
Moffett Field CA 94035, USA

Abstract. This paper summarizes some experiences applying the AT&T Pixel Machine™ to problems of interest in applied computational aerodynamics. These included applications of texture mapping and ray tracing to checking of computational patch geometries. Typical timings for a 964dx Pixel Machine and Sun 3/260 host are given.

[1]This work was supported by NASA Contract NAS 2-11555 while the author was an employess of Sterling Software under contract to the Numerical Aerodynamic Simulation Systems Division at NASA Ames Research Center.

March 21, 1989

Copies of this report are available from:
NAS Applied Research Office
Mail Stop 258-5
NASA Ames Research Center
Moffett Field CA 94035
(415)-694-5197

Experiences with the AT&T Pixel Machine at NAS

Background:

AT&T Pixel Machines (PM) has made available to NAS a "near-production" version of their high-end machine, the 964dx, which has a total of 82 Digital Signal Processing (DSP) chips arranged in an 8x8 array of "pixel nodes," and 2 "pipelines" of 9 DSP's each. Generally speaking, the pixel node array is used for pixel-based operations (like ray-tracing, rendering, compositing, z-buffering, usually acting on 8x8 "footprints" of pixels) and the pipes are used for processing vectors and polygons (very much like SGI's geometry engines). The Pixel machine relies on a host machine, in our case a Sun 3/260, to do non-graphics operations, and hides the graphics operations on the DSP's in graphics libraries.

We were provided with PIClib, which is a graphics library similar to SGI's GL2 and with RAYlib, which is has similar commands, but is dedicated to raytracing. PM has promised a volume rendering library at some indefinite future date.

It is possible to program the chips directly to do other operations. Most notably, ACT (New Mexico) programmed the block tridiagonal inversions from ARC2D onto the pixel nodes and realized 27 "real" megaflops on those parts of the code. Tony Hasegawa has the source code for this. It is quite a job, almost but not quite as bad as writing assembly language, but one does have constantly to check status between the host and the DSP's.

The PM is optimized for pixel-based operations. These occur primarily in the image processing arena, and less so in the vector- and polygon-based computer graphics area. These areas overlap only slightly at present, but that the area of overlap will increase, particularly as processing speeds increase to the point where some of these operations can be handled interactively.

Benchmarks:

Kevin McCabe [personal communication] has reported separately about the performance of the PM on NAS Graphics benchmarks and got results of 2500 polys/sec in the "Optional Faceted Test #7" and 1500 polys/sec in the "Optional Gouraud Test #8." He concludes that these figures are lower by an order of magnitude from other currently available workstations. I found that one gets numbers like these (or better) for texture-mapped polygons as well, i.e. there no penalty for texture mapping.

I had intended to look at three areas of applications, but only got around to two of them. I looked at texture mapping and ray-tracing, both of which are useful in the area of geometry definition and checking. In particular, the texture mapping and ray-tracing capabilities of the PM can be used to check bicubic patch geometries for accuracy.

Texture Mapping:

In texture mapping, one can map a pattern (even a digitized photo) onto bicubic patches or other primitives. For our purposes, a regular stripe or checkerboard is best. This method allows one to check on the location of patch boundaries, look for holes, and the like. It is usually slower than Gouraud shading, but more than an order of magnitude faster than raytracing.

Several different patch objects were texture mappedd, ranging from a single patch to the Martin Newell teapot (28 patches) to significant (1920-patch) components of Chris Atwood's F-16 patch model. While useful for timing purposes, a model with more than about 100 patches produces images where the texture is not clearly visible.

The PM supports the common patch bases (Hermite, Bezier, B-spline and Cardinal-spline) for cubic patches. NURBS (Non-Uniform Rational B-Splines) are not supported as yet. Two "patch precision" parameters (precu, precv) vary between 2 and n. The number of triangles produced is 2*(precu-1)*(precv-1), i.e. roughly as the square of the precision. The drawing time is proportional to the number of polygons, hence proportional to the products of the precisions.

Some timings for the F-16 fuselage are given here. The overall rates are about 5000-6000 lighted (4 sources) texture-mapped triangles per second. The rendering was done in an iteration loop, and averaged over 10 iterations or more.

Texture Mapping timings:

For Chris Atwood's F-16 fuselage (1920) patches 512x512 screen, 40x80 pixel image size

Precision	triangles/patch	triangles	timing	polys/sec
3-3	8	15,360	2.8	5500
5-5	32	61,440	10.2	6100
7-7	72	138,240	28.5	4500
9-9	128	245,760	40.0	6100

Raytracing:

Raytracing can be used to produce a "reflection map" of a patch object. Thus, for example, an aircraft that is being modeled can be made highly reflective (mirror finish) and placed in front of a textured background (vertical stripes are common) and the viewer can be positioned to look at the reflections of the model. This effect is commonly used in the automotive industry to check stampings for defects. This is a particularly good method for checking continuity across patch boundaries, where texture mapping won't work.

There are some standard ray-tracing "benchmarks," most notably those of Eric Haines, but they are not particularly suited to the sorts of uses that we would expect to encounter. The usual models have lots of spherical reflective surfaces, lenses, mirrors, and procedural objects. Things which affect the timing of a ray-tracer are the number of bounding volumes, the image size on the screen (both the whole scene and the sub-components), the number of discreet objects, and the depth of recursion. In the cases noted below, all images are 512x512 pixels, and the standard recursion depth on the PM is 16. (This is rather high, so comparisons with other ray-tracers might be misleading if they are much shallower.)

One thing that I found in the ray tracing work is that judicious use of bounding volumes around disjoint components of objects, can lead to dramatic speedups in raytracing an object (up to a factor of three was observed).

I chose the same two two examples. The first is the Martin-Newell teapot of 28 patches (27, missing one?). The second was the lower fuselage patches from Chris Atwood's F-16 model (1920 patches). This was too many patches for a useful picture, as there were generally many polygons per pixel. These timings are for at least two runs on an unloaded Sun and PM.

For a single patch, a precision of 16 was about enough to cause a single raytracing (patch and two spheres) to take over two hours of CPU time. The longest processing times I tried were of the order of 16 hours. There was a page fault on a single DSP during two of these executions that lasted ten hours or more, and two of them came off without a fault, so the hardware failure rate is significant for "overnight" type runs. Here are the results of ray-tracing a figure with two extra reflecting spheres. At least three runs were conducted for each test.

Raytracing Timings:

Single bicubic Bezier patch, texture -mapped background
2 reflecting spheres, 5 bounding volumes (including one overall) 512x512 pixel image.

Precision	triangles/patch	time (sec)	triangles/sec
3-3	8	11±1	0.72
5-5	32	26±1	1.28
7-7	72	50±2	1.44
9-9	128	85±3	1.51
17-17	512	325±10	1.58
33-33	2048	1375±30	1.49

tea.neu
The Martin Newell Teapot, 27 bicubic Hermite patches, 29 bounding volumes,
2 reflecting spheres, 512x512 screen, image size app 256x256 pixels

Precision	triangles/patch	triangles	sec	triangles/sec
3-3	8	216	8.0±.2	27
5-5	32	864	18.0±.3	48
7-7	72	1944	33.7±.3	58
9-9	128	3456	630.±10	5.5
11-11	200	5400	2080.±60	2.6

fusl.pat
Chris Atwood's F-16 lower fuselage, 1920 bicubic B-spline patches , 1921 bounding volumes,
two reflecting spheres, ray-traced 512x512 screen, image size 100x100 pixels

Precision	triangles/patch	triangles	sec	triangles/sec
3-3	8	15,360	120.±15	128
5-5	32	61,440	990.±30	62
7-7	72	138,240	3300.±50	42

Need for Software, Compatibility, Connectivity &c.

It is a fine thing to obtain new hardware, but its potential is not really tapped until there is also comprehensive and easy-to-use software. Otherwise, we are constantly on the low end of the learning curve, gathering expertise slowly, only to be back at square one when a new machine arrives on the loading dock.

Required software goes beyond the realm of graphics libraries, and extends to functions like metafile support, I/O to hardcopy devices, and "glue" programs of the sort mentioned in the NSF ViSC report. *We never did get a hardcopy off this machine: the labor involved would have been considerable.* The "img" format used in the PM is supposedly proprietary. Can we have a translator? This is especially true in the case of the PM since one of the things it is good for is preparing images with depth-buffering, anti-aliasing, digital compositing, etc. This is not of much use without the capability to read in and use files from other systems.

In order to make serious use of a ray-tracer, it is necessary to have an interactive program to set the scene. The same will be true of volume visualization methods. In the present case, this means that the PIClib code and RAYlib code have to be in the same module, separated by system calls which reprogram the DSP's. I managed to do this; the time to re-load the DSP's is about 10-15 sec per call.

Recommendations to NAS:

Image processing will increasingly overlap with our present practice in computer graphics, particularly if digitized experimental data is to be compared with calculation. The Pixel Machine is undoubtedly useful for non-CFD applications in NASA like LANDSAT imagery interpretation. The PM could be useful in the graphics production area, especially in the production of high-quality still images with good compositing, etc., but it would require a dedicated person to develop software and interfaces for it. It is too expensive to use as a personal workstation, and has interactive capability which, except for texture mapping, is outclassed by current graphics-only workstations. The computing capabilities for small ARC2D segments are impressive, but since usual NAS problems outstrip the capabilities of the largest available supercomputers, it is hard to see what flow problems are usefully done on a workstation. It is certainly worth keeping track of new developments of the PM, especially if good, fast volume rendering can be demonstrated. ❑